THE TERM REWRITING APPROACH TO AUTOMATED THEOREM PROVING

JIEH HSIANG,* HÉLÈNE KIRCHNER, PIERRE LESCANNE, AND MICHAËL RUSINOWITCH

▷ Reasoning about equality has been one of the most challenging problems in automated deduction. The term rewriting method has been one of the most successful approaches for this problem. In this paper, we give a survey of different methods developed in term rewriting for application to automated deduction in various logical systems. ◁

1. INTRODUCTION

Reasoning about equality has been one of the most challenging problems in automated deduction. In the past thirty years, a number of methods have been proposed. In this survey, we give an overview of one of the more successful approaches, the *term rewriting method.*

Term rewriting was first proposed by Evans [38] and Knuth-Bendix [93]. Its original purpose was for generating *canonical term rewriting systems* which can be used as decision procedures for proving the validity of equalities in certain equational theories. With the emergence of equationally specified abstract data types in the late 1970s, term rewriting has gained considerable popularity also as a bridge between programming language theory and program verification. Its application to theorem proving has also been extended to different domains.

The term rewriting approach to theorem proving is unique in several aspects. It is one of the few methods which can be applied to a variety of problem domains. In addition to the validity problem of equational logic, it has also been applied to inductive theorem proving, first-order theorem proving, unification theory, and

*On leave from SUNY at Stony Brook. Research supported in part by NSF grants CCR-8805734. INT-8715231 and CCR-8901322 for the first author and by CNRS for the others.

Address correspondence to J. Hsiang, H. Kirchner, P. Lescanne, or M. Rusinowitch, CRIN-CNRS and INRIA-Lorraine, BP 239, 54506 Vandoeuvre-les-Nancy Cedex, France. E-Mail: hsiang@sbcs.sunysb.edu, {hkirchner, lescanne, rusi}@loria.fr.

Accepted June 1991.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/92/$5.00

even geometry theorem proving. From the operational point of view, term rewriting is a *forward chaining method.* In other words, its main inference mechanism deduces new lemmas from known theorems, rather than reducing the intended problem into subproblems and trying to solve them separately. Forward chaining methods are usually not efficient due to the large number of theorems produced which makes the search space unmanageable. (As a tradeoff, forward chaining methods usually do not require backtracking, which is necessary in most backchaining methods.) In fact, term rewriting is one of the very few successful forward chaining methods. The problem of space explosion is handled in term rewriting through two techniques: a notion of *critical pairs*, which tries to find only "useful" lemmas, and more importantly, a notion of *simplification.* Basically, simplification replaces existing data by those which are logically equivalent but "smaller" according to some well-founded ordering. Since the ordering is well-founded, simplification cannot go on indefinitely.

It has been demonstrated through various implementations and experiments that simplification is indeed an effective way of controlling the search space. In fact, it is fair to say that term rewriting is presently the best approach to the theorem proving problem of equational logic. In addition to finding complete sets of reduction rules, other notable problems have been solved using term rewriting including the one-axiom group theory [106], the commutativity problem of rings [137], the ternary Boolean algebra problems [43], and the Moufang identities of alternative rings [2].

This paper is concerned with the use of term rewriting in automated theorem proving for various logical systems and is organized according to the generality of the logical languages. In Section 2, we present term rewriting methods for purely equational logic. In addition to Knuth-Bendix completion, we describe its extensions to unfailing completion, inductive equational theorem proving, as well as rewriting modulo a set of equalities. Different term rewriting methods for Horn logic with the equality predicate are presented in Section 3. Finally in Section 4, we present term rewriting methods for the full first-order logic with equality. Some research issues that need to be addressed are pointed out in the conclusion.

This brief survey reflects more or less our participation in the application of term rewriting to automated theorem proving. It is an attempt to give a progressive and homogeneous presentation of this topic. To achieve this goal, in all the logical systems considered, theorem proving methods are presented using a common formalism of inference rules. This choice is intended to emphasize the connections and common features among the different theorem proving systems. This presentation allows a higher level of abstraction based on which further refinements of the strategies can be made more effectively. The paper by no means exhausts all the work in the rewriting and theorem proving areas. More recent work on these subjects can be found in the conferences of *Automated Deduction* and *Rewriting Techniques and Applications*, as well as journals in artificial intelligence and theoretical computer science such as the *Journal of Symbolic Computation* and the *Journal of Automated Reasoning.*

2. EQUATIONAL LOGIC

In equational logic [55, 131, 139, 140], a formula is a universally quantified equality on terms. *Equational theorems* are deduced from a set A of *equational axioms* by

an inference rule called *replacement of equals by equals*. It consists of replacing, in a term t, an instance $\sigma(l)$ of a left-hand side of an axiom $(l = r)$ by the same instance of its right-hand side $\sigma(r)$, thus producing another term s. This is denoted by $t \leftrightarrow_A s$, while $\overset{+}{\leftrightarrow}_A$ and $\overset{*}{\leftrightarrow}_A$ denote respectively the transitive and the reflexive transitive closure of this replacement relation. Such proofs are called *equational proofs*.

Terms are built on a signature $\mathcal{F}$ that gives the set of operators with their arity and a set of variables $\mathcal{X}$. The set of terms is denoted $\mathcal{T}(\mathcal{F},\mathcal{X})$, while the set of ground terms (without variables) is denoted $\mathcal{T}(\mathcal{F})$. A model of a set of axioms A is an algebra satisfying A. From an important completeness theorem, due to Birkhoff [17], it follows that the inference rule of replacing equals by equals is sufficient to prove equalities valid in all models of A. Such equalities are denoted by $t =_A s$.

Example 2.1. Consider the signature $\mathcal{F}$ composed of three opeators e, i, $*$ taking respectively 0, 1 and 2 arguments. The set {0, 1} together with the operations $e_{\mathcal{A}} = 0$, $i_{\mathcal{A}}(0) = 0$, $i_{\mathcal{A}}(1) = 1$, $0 *_{\mathcal{A}} 0 = 1 *_{\mathcal{A}} 1 = 0$, $1 *_{\mathcal{A}} 0 = 0 *_{\mathcal{A}} 1 = 1$ is an algebra $\mathcal{A}$ of signature $\mathcal{F}$. Consider the following set, called ***Group*** of axioms for groups.

$$x * e = x$$

$$x * i(x) = e$$

$$(x * y) * z = x * (y * z).$$

The previous algebra $\mathcal{A}$ is a model of ***Group***.

An equational proof of $e * x = x$ in groups is given below:

$$\begin{aligned} e * x &= e * (x * e) = e * (x * (i(x) * i(i(x)))) = e * ((x * i(x)) * i(i(x))) \\ &= e * (e * i(i(x))) = (e * e) * i(i(x)) = e * i(i(x)) \\ &= (x * i(x)) * i(i(x)) = x * (i(x) * i(i(x))) = x * e = x. \end{aligned}$$

2.1. Rewriting

Equational proofs of equational theorems are not easy to deduce. It is difficult to find a strategy for selecting and orienting axioms. A simple idea is always to apply the equalities in the same direction. Such a directed equality is called a *rewrite rule*, denoted by replacing the equal sign $=$, by an arrow $\rightarrow$. A (finite or infinite) set of rewrite rules is a *rewrite system*.

Example 2.2. For instance,

$$x * e \rightarrow x$$

$$x * i(x) \rightarrow e$$

$$(x * y) * z \rightarrow x * (y * z)$$

is a rewrite system.

Rewriting a term with a rewrite system R consists in replacing a subterm which matches a left-hand side of a rewrite rule by the right-hand side whose variables are bound to values computed by the matching algorithm. This relation is denoted by $\rightarrow_R$. Iterating this process is called *reducing*. If two terms can be reduced to a

same one, a special equational proof is obtained, called a *rewrite proof*. A term which cannot be rewritten is said to be in *normal form*.

Example 2.3. The rewrite rules of Example 2.2 are used in the equational proof of Example 2.1 as follows:

$$e * x \leftarrow e * (x * e) \leftarrow e * (x * (i(x) * i(i(x)))) \leftarrow e * ((x * i(x)) * i(i(x)))$$
$$\rightarrow e * (e * i(i(x))) \leftarrow (e * e) * i(i(x)) \rightarrow e * i(i(x))$$
$$\leftarrow (x * i(x)) * i(i(x)) \rightarrow x * (i(x) * i(i(x))) \rightarrow x * e \rightarrow x.$$

Obviously it is not a rewrite proof. There are peaks, i.e., terms from which issue two sequences of rewritings, and valleys, i.e., terms where rewriting is not applied any more. Such terms, for instance $e * x$, $e * (e * i(i(x)))$, $e * i(i(x))$ and x are in normal form for the rewrite system.

2.2. Termination

The termination problem is to prove that all rewriting sequences issued from any term will terminate. In this section, we only briefly sketch the problem and its solution; a good and complete survey can be found in [33].

A rewrite system is said to be *terminating* if the relation $\rightarrow_R$ is well founded. Termination is in general undecidable and sufficient conditions for ensuring it have been developed. Classical methods are based on orderings on the term algebra. The key idea is to show that a specific well-founded partial ordering contains the intended rewrite relation. If such an ordering exists, clearly the rewrite relation terminates. However, instead of proving the inclusion on the whole rewriting relation, one usually localizes the proof, by proving that only the well-founded relation contains the rewrite rules. By means of some stability properties, this is extended to a termination proof of the whole rewriting relation. Basically, a "good candidate" has to satisfy two properties: stability by substitution of terms to variables (also called *full invariance*) and stability by the operations of the term algebra (also called *replacement* property).

Such a well-founded ordering is called a *reduction ordering* [32, 33, 101]. *Simplification orderings* [30] form an important family of reduction orderings. They are such that any subterm of a term is smaller than the whole term. This property and stability by substitutions and operators are enough for ensuring the well-foundedness of the ordering.

There are essentially two types of reduction orderings on terms for proving termination. The first one, called *syntactical*, provides the ordering via a careful analysis of the structure of the terms. Among these orderings are the *recursive path ordering* [30], the *lexicographic path ordering* [77] and the *recursive decomposition ordering* [75, 108]. These orderings are convenient since they are based on a concept of precedence which is somewhat natural, especially in the context of abstract data types. In addition they enjoy a property called *incrementality* that enables one to build the proof of termination step by step, in the case of an incremental set of rewrite rules. Another family contains the *semantical orderings*, that interpret the terms in another structure where a well-founded ordering is known. For such a purpose, two common, well-ordered sets are the natural numbers and the terms ordered by a syntactical ordering. The first choice enables

one to consider functions over natural numbers which are stable by substitutions. The most frequently used are polynomials [28, 102]. If the set of terms ordered by a syntactical ordering is chosen as a target for the "semantics," one transforms the rewrite system into another one [7, 14]. The Knuth-Bendix ordering presented with the completion procedure in the original paper [93] combines a semantical ordering (polynomial interpretation of degree 1) and a syntactical ordering.

2.3. *Completion and Equational Proofs by Rewriting*

A property required for a rewrite system is the uniqueness of the normal form for any term. This is crucial for any application where computing normal forms must be independent of the strategy of rule application. Uniqueness of the normal form is implied by another property called *confluence*. A rewrite system is confluent when two rewrite sequences beginning from the same term can always be extended to end with the same term. Although undecidable in general, confluence is decidable for terminating finite rewrite systems.

Confluence (which may be defined as an abstract property of relations, not specifically of rewriting relations) is equivalent to the *Church-Rosser property* that gives the relation between equational proofs and rewrite proofs: given a Church-Rosser rewrite system, every equational theorem has a rewrite proof. Assuming termination, confluence is equivalent to local confluence [113], itself equivalent to the convergence of *critical pairs* [66, 93]. Critical pairs come from overlapping applications of two rewrite rules.

Let R be a set of rewrite rules. By $s[t]$, we mean that the nonvariable term t occurs as a subterm in the term s. The set $CP(R)$ of critical pairs of R is computed by unifying any left-hand side u with a nonvariable subterm u' of another left-hand side $l[u']$.[1] If σ is the *most general unifier* of u and u' [128] (denoted $mgu(u, u')$ for short), the term $\sigma(l)$ can be rewritten in two different ways that produce a critical pair. More formally:

$$CP(R) = \{\sigma(l[v] = r) \mid \text{there exist } (l[u'] \to r) \in R, (u \to v) \in R \text{ such that } \sigma = mgu(u', u)\}.$$

Example 2.4. Equation $(x * z = x * (e * z))$ is a critical pair of the system given in Example 2.2. It is obtained from the term $(x * e) * z$, by rewriting it by the first and the third rule:

$$x * z \leftarrow (x * e) * z \to x * (e * z).$$

A *completion* procedure is aimed at building a Church-Rosser and terminating rewrite system from a set of equalities. Completion computes critical pairs, orients equalities into rewrite rules and keeps terms in normal form for the current set of rewrite rules.

Following [6, 10], completion is described through inference rules with a fair search plan, which transform a set of equalities E and a set of rewrite rules R. They are given below. *Deduce* adds to E equational consequences of R obtained

[1] We always assume that rules have disjoint sets of variables. Note also that l may be a renaming of u.

by critical pairs computation. *Orient* transforms an equality of E into a rewrite rule, taking into account a given reduction ordering in order to ensure termination. *Delete* and *Simplify* respectively removes and reduces equalities in E. *Compose* and *Collapse* reduce the terms in the rewrite rules of R When a left-hand side is reduced, we may need to reconsider the orientation, so the obtained equality goes back to E. In addition each rule can be reduced only by another rule which is smaller in some sense. This is the reason for the condition in *Collapse*.

Deduce:	$E;\ R$	$\vdash E \cup \{s = t\};\ R$	if $(s = t) \in CP(R)$
Orient:	$E \cup \{s = t\};\ R$	$\vdash E;\ R \cup \{s \to t\}$	if $s > t$
Delete:	$E \cup \{s = s\};\ R$	$\vdash E;\ R$	
Simplify:	$E \cup \{s = t\};\ R$	$\vdash E \cup \{s = u\};\ R$	if $t \to_R u$
Compose:	$E;\ R \cup \{s \to t\}$	$\vdash E;\ R \cup \{s \to u\}$	if $t \to_R u$
Collapse:	$E;\ R \cup \{s \to t\}$	$\vdash E \cup \{u = t\};\ R$	if $s \to_R u$ by a rule $l \to r \in R$ with $s \succ l$,

where $>$ is a reduction ordering and $\succ$ is the *encompassment ordering on terms*.[2]

Completion can be understood as a proof simplification process, where each inference rule decreases the complexity of some equational proofs. The minimal (i.e., less complex) proofs in this setting are the rewrite proofs.

Completion is initialized with a given set of equalities in E and an empty set of rules in R. It has three possible outcomes: it terminates, fails on an unorientable equality, or diverges, that is, infinitely generates many new rules. When the completion ends up with an empty set of equalities in E and a Church-Rosser and terminating rewrite system in R, the validity of an equational theorem is decidable by reducing both terms to their normal forms and by checking the syntactic equality of the results.

Example 2.5. The following rewrite system

$$
\begin{aligned}
x * e &\to x \\
e * x &\to x \\
x * i(x) &\to e \\
i(x) * x &\to e \\
i(e) &\to e \\
i(i(x)) &\to e \\
i(x * y) &\to i(y) * i(x) \\
(x * y) * z &\to x * (y * z) \\
x * (i(x) * y) &\to y \\
i(x) * (x * y) &\to y
\end{aligned}
$$

can be obtained by completion from Groups.

Even when the process diverges, completion can nevertheless semi-decide validity of an equational theorem [67]. If a rewrite system produced at some step by the

[2] $s \succ l$ means there exists a subterm of s which is an instance of l and not conversely.

completion procedure reduces both terms of the theorem to a same one, then the theorem is valid.

Completion procedures have been implemented in different systems, such as KB [39], REVE [41, 106], RRL [87], TRSPEC [4], ERIL [37], METIS [116] and ORME [107] (see [57] for a survey and [34] for various applications). Experiments with such systems made clear that very often too many critical pairs are computed. This source of inefficiency motivated works on critical pairs criteria [6]. A critical pair criterion characterizes peaks for which a less complex proof can be found. Criteria such as *blocking* [104, 134], *connectedness* [98, 143] or *compositeness* [83, 147] allow dropping some unnecessary computations of critical pairs.

A historical survey on completion procedures can be found in [26]. The first completion procedure in equational logic is due to Knuth and Bendix [93], while Buchberger proposed almost at the same time a similar procedure for generating Gröbner bases for polynomial ideals [25]. A Gröbner basis of a finitely generated ideal in a polynomial ring is a set of polynomials which can be used as rewrite rules to compute normal forms. A polynomial belongs to this ideal if and only if its normal form is 0. These studies naturally have strong connections with term rewriting [26, 142]. In addition to term rewriting and polynomial rewriting, much work has also been devoted to word rewriting (Thue systems [19]).

2.4. *Unfailing Completion and Proof by Contradiction*

Completion procedures can abort on an equality that cannot be oriented into a terminating rule by the inference rule *Orient*. The idea of extending completion by computing equational consequences of nonorientable equalities can be traced back to [24, 100]. In 1985, the notion of an *unfailing completion* was proposed: in this framework, orientable instances of an equality are used to perform reductions, even if the equality itself is not orientable. For instance, the commutativity axiom $(x * y = y * x)$ may have an instance $(f(x) * x = x * f(x))$ that can be oriented with a lexicographic path ordering. Such an instance reduces the term $(f(a) * a)$ into $(a * f(a))$. This method requires a reduction ordering $>$ which can be extended to an ordering total on *ground terms*, i.e., terms without variables. Polynomial orderings mentioned in Section 2.2 and the Knuth-Bendix ordering satisfy this requirement. Also any total precedence on operators can be used to define a lexicographic path ordering total on ground terms. This restriction can be somewhat relaxed, since the reduction ordering needs only to be total on ground terms which are equivalent with respect to the considered equational theory, as proved in [6].

Given a set of equalities E, $E^>$ denotes the set of all orientable instances of equalities in E:

$$E^> = \{\sigma(g) \to \sigma(d) \mid (g = d) \in E,\ \sigma(g) > \sigma(d)\}.$$

The *ordered rewriting* relation is just rewriting with orientable instances. Note that $s \to_{E^>} t$ implies $s \leftrightarrow_E t$ and $s > t$.

Deduction of new equalities needs the definition of *ordered critical pairs* between two equalities of E, obtained by unifying one term of an equality with a nonvariable subterm of the other one. Such deduced equalities must cover all cases of peaks for the ordered rewriting, which elimintes some unifying substitutions.

The set $OCP(E)$ of ordered critical pairs of E is precisely defined as follows:

$$OCP(E) = \{\sigma(l[v] = r) \mid \text{there exist } (l[u'] = r) \in E, (u = v) \in E \text{ such that } \sigma = mgu(u', u) \text{ and } \sigma(l[u']) \not\succeq \sigma(r), \sigma(u) \not\succeq \sigma(v)\}.$$

As for completion, a set of inference rules is given for unfailing completion. Again *Deduce* adds in E equational consequences of E obtained by ordered critical pair computation. *Delete* removes trivial equalities in E. *Collapse* reduces the terms in the equalities. Each equality can be reduced only by another one which is smaller in some sense. This is the reason of the condition in *Collapse*.

Deduce: $E \vdash E \cup \{s = t\}$ if $(s = t) \in OCP(E)$

Delete: $E \cup \{s = s\} \vdash E$

Collapse: $E \cup \{s = t\} \vdash E \cup \{s = u\}$ if $t \to_{E_\succ} u$ by an equality $l = r \in E$ with t or $t = l$ and $s > r, \times$

where $\succ$ is again the encompassment ordering.

The unfailing completion procedure has only two possible outcomes: either it generates a finite *ordered rewrite system* $(E, >)$ or it diverges. In the first case, it provides a decision procedure for validity of any equational theorem. In the second case, it provides a semi-decision procedure [6, 11, 65, 129]. Unfailing completion is implemented in SbREVE [63, 112], RRL, METIS, ORME and TRSPEC.

Example 2.6. The unfailing completion applied to the following axioms, with the lexicographic path ordering (lpo):

$$(x * y) * (z * w) = (x * z) * (y * w)$$
$$(x * y) * x = x$$

gives an ordered rewrite system with the following equalities:

$$(x * y) * x = x$$
$$z * (y * x) = z * x$$
$$((x * y) * z) * w = x * w$$
$$(x * y) * w = (x * z) * w.$$

For proving a universally quantified equational theorem like

$$(x_1 * (y_1 * u_1)) * v_1 = (x_1 * x_1) * v_1,$$

the ordering is extended to take into account the variables of the theorem. We may consider now lpo with the following precedence: $* > x_1 > y_1 > u_1 > v_1$.[3] Each side of the theorem matches with the left-hand side of the fourth rule. Replacing z in the right-hand side of this rule by the smallest symbol v_1, two ordered rewriting steps can be applied. The normal form of each side of the theorem is: $(x_1 * v_1) * v_1$. This achieves the proof.

[3] Since the target theorem is only used in the reduction process but not in the deduction process, its variables will never be instantiated and may be treated as constants. This justifies including them specifically in the ordering.

Another use of unfailing completion is a familiar method for mathematical proof, namely proof by contradiction: assume the negation of the formula to be proved and derive a contradiction. To refute this negation, the unfailing completion applies the previous set of inference rules until it generates a contradiction [10, 11]. The unfailing completion has been proved refutationally complete [6, 11, 65, 129]. The next example illustrates this refutational method on a satisfiability proof.

Example 2.7. For proving that the existentially quantified formula

$$\exists x, y, z, (x - y) + z = x$$

holds in the theory described by the following axioms:

$$(x - y) + z = (x + z) - y$$

$$(x + y) - y = x,$$

the problem is encoded by the two following equalities:

$$eq(x, x) = true$$

$$eq((x - y) + z, x) = false.$$

Unfailing completion generates then the equality eq(x, x) = false whose critical pair with eq(x, x) = true yields a contradiction true = false.

2.5. *Proving Inductive Theorems*

Some proofs of properties on classical data structures, such as integers, require *induction.* An *inductive theorem* is an equality that is true in the initial model of the axioms.

Example 2.8. Natural numbers are formed with two operators: a constant 0 and a unary operator *succ* that represents the successor of any natural number. Addition is defined by the operator + and two axioms:

$$x + 0 = x$$

$$x + succ(y) = succ(x + y).$$

The associativity of + is expressed by the equality

$$(x + y) + z = x + (y + z).$$

This equality cannot be deduced using replacement of equals by equals, although it is true for natural numbers.

The principle of a *proof by consistency* is to assume the validity of the intended inductive theorem and show that there is no contradiction [82]. More precisely, the equality e is valid in the initial model of a set of axioms A iff A and $A \cup \{e\}$ have the same initial model. (See [74] for a history of proof by consistency.)

When there exists a confluent and terminating rewrite system for A, the method can take advantage of the existence of free constructors, characterized as a subset $\mathscr{C}$ of $\mathscr{F}$ such that $\mathscr{T}(\mathscr{C})$ is exactly the set of normal forms of $\mathscr{T}(\mathscr{F})$. In this case, every nonconstructor operator must be *completely defined* [31, 94, 105, 141], that is any ground term containing this operator is reducible. The latter property is

decidable for a confluent and terminating rewrite system. Once it is checked, the completion of the rewrite system enriched by the theorem to be proved is invoked. If no relation (no critical pair) between two distinct ground terms in normal form is found, the inductive theorem is true. Otherwise, it is false.

The idea of using a completion procedure to prove inductive theorems was first developed in [52, 68, 103, 111]. Further progress has been made with the introduction of the *ground reducibility* property for a term (also called *inductive reducibility*), meaning that any instantiation of its variables by ground terms is reducible [74]. If the left-hand side of each rule added during the completion satisfies this property, the normal forms of ground terms are preserved. This provides another method for inductive proofs in the initial algebra that avoids the complete definition property. Ground reducibility is decidable for finite rewrite systems [86, 121]. However deciding ground reducibility is in exponential time even for left-linear rules. Algorithms for deciding ground reducibility in the case of left-linear rules have been given, for instance in [29, 74, 86, 97].

An alternative to the ground reducibility approach is the *test set* method proposed by [85]. A test set for a rewrite system R gives a finite description of irreducible ground terms. This concept is also applied to check the complete definition property.

A further improvement is to use an *unfailing proof by consistency* [5, 6] which avoids failure due to a nonorientable inductive theorem like commutativity. This method is *refutationally complete* in that it refutes any equality which is not an inductive theorem. For that, it detects any *provably inconsistent* equality, i.e., any equality $(s = t)$ which satisfies either $s > t$ (in the reduction ordering $>$) and s is not inductively reducible, or $(s = t)$ is not inductively reducible (i.e., neither s nor t is inductively reducible).

Inference rules for proof by consistency are given in [5]. Let R be a confluent and terminating rewrite system that describes the equational theory. Let C be any set of conjectures to be proved and L any set of inductive lemmas. Inductive lemmas are equational theorems $(s = t)$ such that $\sigma(s) =_R \sigma(t)$ for any ground substitution σ, which is also denoted by $(s =_{ind(R)} t)$. *Deduce* adds in C new conjectures to be proved and obtained by computation of critical pairs in $CP(R, C)$, the set of critical pairs obtained by superposition of rules in R into conjectures in C. Once a conjecture has been proved valid, it can be deleted from C, using *Delete*, then added to L using *Induce*. The *Induce* inference rule allows also the introduction of inductive lemmas given by the user or produced by another system. *Simplify* reduces conjectures, using either rules of R, lemmas of L or other smaller conjectures of C. Finally, *Refute* produces a disproof when a provably inconsistent conjecture is detected.

Deduce:	$L; C \vdash L; C \cup \{s = t\}$	if $(s = t) \in CP(R, C)$
Delete:	$L; C \cup \{s = t\} \vdash L; C$	if $s =_{ind(R)} t$
Induce:	$L; C \vdash L \cup \{s = t\}; C$	if $s =_{ind(R)} t$
Simplify:	$L; C \cup \{s = t\} \vdash L; C \cup \{u = t\}$	if $s > u$ *and either* $s \overset{+}{\leftrightarrow}_{R \cup L} u$ *or else* $s \leftrightarrow_C u$
	$\times$	*by* $(v = w)$ *s.t.* s *and* $v > w$
Refute:	$L; C \cup \{s = t\} \vdash$ *Disproof*	if $(s = t)$ *is provably inconsistent.*

Here again $\succ$ is the encompassment ordering on terms.

Example 2.9. In Example 2.8, the operator + is completely defined by the Church-Rosser and terminating rewrite system:

$$x + 0 \to x$$
$$x + succ(y) \to succ(x + y).$$

The associativity of +

$$(x + y) + z = x + (y + z)$$

can be added as a new equality and proved by consistency as follows. Choose the ordering such that $x + (y + z) > (x + y) + z$. Since $x + (y + z)$ is ground reducible, the equality $(x + y) + z = x + (y + z)$ is not provably inconsistent. Then it is enough to superpose R on $x + (y + z)$. This gives, by unifying $(y + z)$ and $x + 0$, the critical pair $x + y = (x + y) + 0$, that is simplified into $x + y = x + y$ and eliminated. Since no provable inconsistency has been generated, this achieves the proof of associativity.

Following [52, 103], all these methods are called *inductionless induction* methods. They are often inefficient and may not terminate. Several concepts have been proposed to limit the number of superpositions to perform: *covering sets* [5], *cover sets* [148], or *complete positions* [44, 99]. These notions actually correspond to the selection of an induction term in usual inductive proof methods, used for instance in AFFIRM [110], in NQTHP [22] or in LP [51].

Recently, another approach has been proposed in [123] to address inductive reasoning. Proving an inductive conjecture amounts to proving some specific instances. Each one is simplified by either a rule or a smaller instance of the conjecture until an identity is obtained. In [123], the relationship between this new method and the inductionless induction procedures is clarified.

2.6. *Equational Rewriting: Built-In Equality*

Some equalities cannot be oriented into rules, but can be built into the rewriting process itself. This requires the elaboration of a new abstract concept, namely the notion of *rewriting modulo a set of equalities* in which the matching takes into account nonoriented equalities. Adequate notions of *confluence and coherence modulo* a set of equalities [66, 70] have been defined for this kind of rewriting relation. A *class rewrite system* is defined by a set R of rewrite rules and a set A of equalities. It is *Church-Rosser modulo A* if any equational theorem deduced from $R \cup A$ has a rewrite proof using rewriting in equivalence classes modulo A, defined as the compound relation $=_A \circ \to_R \circ =_A$. The class rewrite system is *terminating modulo A* if $=_A \circ \to_R \circ =_A$ terminates. Equational completion procedures generalize the standard case [9, 73, 104, 120], but they must be carefully studied to minimize sources of inefficiency.

Typical examples handled by such a method are theories with associativity and commutativity axioms [69, 90, 120].

Example 2.10. Starting from the two axioms

$$x + 0 = x$$
$$x + (-x) = 0$$

and the fact that + is associative and commutative, an equational completion procedure generates an equational rewrite system for abelian groups, given by the following sets of rewrite rules and equalities:

$$
\begin{aligned}
x+0 &\to x \\
x+(0+y) &\to x+y \\
x+(-x) &\to 0 \\
x+((-x)+y) &\to y \\
--x &\to x \\
-0 &\to 0 \\
-(x+y) &\to (-x)+(-y) \\
x+y &= y+x \\
(x+y)+z &= x+(y+z).
\end{aligned}
$$

Equational completion assumes the existence of an equational unification algorithm for the subtheory defined by the nonoriented axioms A.

2.7. *Equational Unification*

Unification is related to the problem of satisfiability in equational theories. The goal is to find substituions σ, or values for the variables in an equation $(s = t)$, such that the two substituted terms $\sigma(s)$ and $\sigma(t)$ are in the same equivalence class modulo the axioms A of the theory. Although the number of substitutions may be infinite, we are usually only interested in a generating subset, called a *complete set of unifiers*. Complete sets of unifiers are important when considering the completeness of theorem-proving strategies, since they provide a way to cover all the possible consequences of a deduction step.

When the set of axioms is empty, a unifier σ of s and t is the most general unifier if for every unifier θ of s and t, there exists a substitution ρ such that $\theta = \rho \circ \sigma$ on the variables of s and t. The most general unifier is unique up to renaming of variables [128] and the minimal complete set of unifiers of two given terms is either empty or a singleton.

This definition generalizes to equational unification: a set $CSU(s, t, A)$ of A-unifiers of s and t is a complete set if for every unifier θ of s and t, there exists a substitution ρ in $CSU(s, t, A)$ such that $\theta =_A \rho \circ \sigma$ on the variables of s and t. When the axiom set is not empty, the minimal complete sets of unifiers can be either finite or infinite.

Example 2.11. In the associative theory, the simple equation $a * x = x * a$ has the following infinite minimal complete set of unifiers:

$$
\begin{aligned}
&x \mapsto a \\
&x \mapsto a * a \\
&x \mapsto a * a * a \\
&\quad\vdots
\end{aligned}
$$

Since the pioneering work of Plotkin [122] and Stickel [136], the field has grown. Methods and tools for the design of unification procedures are presented in [49, 88, 135] for the specific problem of equational unification. In addition to improving already-known unification algorithms and discovering new ones, significant progress

has been made in regard to the combination of unification algorithms. Another improvement was to provide completion-based tools allowing the automatic computation of a (potentially infinite) unification procedure from the axioms of the theory [89]. More references can be found in surveys [71, 133].

The *narrowing process* is a general unification method that yields complete sets of unifiers, provided that the theory can be presented by a terminating and confluent rewrite system [40, 69, 72]. *Narrowing* an equation consists of replacing a nonvariable subterm which unifies with a left-hand side of a rewrite rule by the right-hand side, and instantiating the result with the computed most general unifier.[4] This process is iterated until an equation is found such that both sides are unifiable. Then the composition of their most general unifier with all the substitutions computed by narrowing yields a unifier in the equational theory. The narrowing process consists in building all the possible narrowing derivations starting from the equation to be solved. This method is incremental in that it allows building, from a unification algorithm in a theory A, a unification procedure for a theory $R \cup A$, provided the class rewrite system defined by R and A is Church-Rosser and terminating modulo A [72]. The drawback of such a general method is that it very often diverges and several attempts have been made to restrict the size of the narrowing derivation tree [36, 69, 114, 125, 126, 145].

An interesting restricted problem is the *rigid* equational unification problem that has the advantage of being decidable (but NP-complete) [46]. Given a finite set A of equalities, a *rigid A-unifier* of two terms s and t is a substitution σ such that $\sigma(s)$ and $\sigma(t)$ can be proved equal using the set $\sigma(A)$ as a set of ground equalities. The design of a decision procedure for rigid A-unification needs first the computation of a reduced set of rewrite rules *rigid equivalent* to the given finite set of equalities A. Then the computation of complete sets of rigid A-unifiers is done by a process which is very similar to equational narrowing [48].

3. HORN CLAUSE LOGIC WITH EQUALITY AND CONDITIONAL REWRITING

A way to improve the expressiveness of the logic is to introduce conditional rules. Several approaches have been developed and more effort is needed to understand their relationship.

3.1. Conditional Rewriting: An Algebraic Approach

The formulas being considered are *conditional equalities*, written "$c \Rightarrow s = t$," where c is a conjunction of equalities. Its meaning is that s and t are equal *if* the condition c is satisfied. Models are again algebras satisfying a given set of conditional equalities. In a *conditional rewrite rule*, the equality is oriented and denoted as "$c \Rightarrow s \to t$." A formula to be proved may be either an equality or a conditional equality.

Example 3.1. Assume given an axiomatization of Boolean algebras with function symbols $\wedge$ and $\vee$ which denote the logical connectives 'and' and 'or'. Assume also given a total ordering $\leq$ and its associated strict ordering $<$. The operation inf

[4] The equation and the rule are assumed to have disjoint sets of variables.

that computes the minimum of two elements with respect to $\leq$ is described with the following conditional axioms:

$$eq(x,x) = true$$
$$(x \leq y) \wedge (y \leq x) = eq(x,y)$$
$$(x < y) \wedge (y \leq x) = false$$
$$(x < y) \wedge (y < x) = false$$
$$(x < y) \vee (y \leq x) = true$$
$$(x < y) = true \Rightarrow inf(x,y) = x$$
$$(y \leq x) = true \Rightarrow inf(x,y) = y.$$

Following earlier work, a Birkhoff-like theorem establishes the completeness of conditional replacement of equals by equals [132] (see also [15, 23]).

Two approaches have been considered in conditional term rewriting. In *recursive rewriting* [78, 79], the condition of a rule is evaluated first and the rule is applied only when the condition holds. Thus conditional rules introduce the additional complexity of recursively evaluating the conditions, which gives rise to a new termination problem. Knowing whether a term is reducible or not is undecidable [78]. To handle this problem, Kaplan in [79] introduced the notion of *simplifying systems*. In a simplifying system every left-hand side of a rule is greater than both its right-hand side and its condition in some simplification ordering. Since such an ordering is well founded, every term can be brought to a normal form via a simplifying system. Simplifying systems have been generalized to *reductive systems* [76] and to *decreasing systems* [35].

In *contextual rewriting* [124] rules may be applied to terms without evaluating the condition, which is appended to the term as a context. In order to ensure correctness of such a reduction, a term with a complementary context must also be created. This approach is subject to an obvious nontermination risk; in general, the contextual part of a term may grow unbounded during a sequence of reductions.

As in the equational case, the Church-Rosser property is equivalent to confluence of the recursive conditional rewriting relation, and confluence is equivalent to local confluence. This last property is in turn equivalent to convergence of conditional critical pairs, provided that the system is decreasing [35]. However, the problem of determining whether critical pairs always converge still remains.

An approach based on contextual rewriting is implemented in Reveur4 [21], a version of REVE for conditional rewriting. The idea is to use case splitting on the condition in order to partition the set of critical pairs into different subsets. It is then checked whether the contexts are unsatisfiable or whether the terms are convergent without additional hypotheses [81]. In this approach, the class of models is limited to the subclass of algebras where case reasoning is valid. In these models, the Boolean part is isomorphic to the classical two-element Boolean algebra [20].

Example 3.2. The set of conditional axioms in Example 3.1 can be oriented from left to right and gives a Church-Rosser and terminating set of conditional rewrite rules.

The equality $(\inf(x, y) = \inf(y, x))$ can be proved by contextual rewriting: in the context $(x < y)$, both terms reduce to x, in the context $(y < x)$, both terms reduce

to y, and in the context eq(x, y), both terms reduce respectively to x and y which are equal.

However, contextual rewriting considers only restrictive conditional rules: the conditions need to be Boolean equations whose right-hand sides belong to the set {*true*, *false*}. Moreover, several additional hypotheses are required to force the models to be consistent extensions of the two-element Boolean algebra.

In the case of recursive rewriting, the main problem is the treatment of nonsimplifying equalities in which the condition is more complex than the conclusion. An interesting approach to overcome this problem is to superpose rules on the condition in order to enumerate its solutions [95]. This process translates the nonsimplifying equality into a set of simplifying rules. Unfortunately this set is infinite in general and additional techniques must be used to ensure termination of completion [50].

3.2. A Refutational Approach

A conditional rewrite rule where the condition c is a conjunction of positive literals is also known as a definite Horn clause. This remark gives rise to a different approach based on a set of inference rules which is refutationally complete for Horn clause logic with equality [95]. These rules are direct extensions of unfailing completion. In all deduction mechanisms presented so far, term replacement only took place within the larger member of an equality and a term was never replaced by a larger one. Similarly, for Horn clauses, any term replacement is performed only into the largest literals in a clause using the largest equality literal of a clause.[5]

In order to simplify the presentation, it is assumed that " $=$ " is the only predicate which occurs within the encountered clauses. The empty clause is denoted by the only symbol " $\Rightarrow$." In the following, c and g are conjunctions of equalities and e is an equality.

The notion of critical pair can be extended to conditional equalities. Let E be a set of Horn clauses. By $C[t]$ we mean that the term t occurs as a nonvariable subterm in the clause C. A clause can be viewed as a set of literals, and the set-containment relation on clauses is denoted by $\subseteq$.

The set of *conditional critical pairs* of E is obtained by unifying a left-hand side u of a conditional equality with a nonvariable subterm u' of another left-hand side $l[u']$.

$$
\begin{aligned}
CCP(E) = \{\sigma(c \wedge q \Rightarrow l[v] = r) \mid\ & \\
& \text{there exist } (c \Rightarrow l[u'] = r) \in E, (q \Rightarrow u = v) \in E \\
& \text{such that } \sigma = mgu(u', u) \\
& \text{and } \sigma(l[u'] = r) \not\subseteq \sigma(c), \sigma(l[u']) \not\subseteq \sigma(r), \\
& \text{and } \sigma(u) \not\subseteq \sigma(q), \sigma(u) \not\subseteq \sigma(v)\}.
\end{aligned}
$$

We also need to partially solve conditions of equalities when they are more complex than the conclusion. This motivates the introduction of the set of *conditional narrowings* computed from E, in which the left-hand side u of a conditional

[5] Literals and clauses are compared with an extension of a simplification ordering (see [130]).

equality is unified with a nonvariable subterm u' occurring in the condition of another one.

$$CN(E) = \{\sigma(c \wedge q \wedge l[v] = r \Rightarrow e) \mid$$
$$\text{there exist } (c \wedge l[u'] = r \Rightarrow e) \in E, (q \Rightarrow u = v) \in E$$
$$\text{such that } \sigma = mgu(u', u)$$
$$\text{and } \sigma(l[u'] = r) \not< \sigma(c), \sigma(l[u']) \not\leq \sigma(r), \sigma(l[u'] = r) \not\leq \sigma(e),$$
$$\text{and } \sigma(u) \not\leq \sigma(q), \sigma(u) \not\leq \sigma(v)\}.$$

In the above definition, we also allow e to be missing. The formula $(c \wedge l[u'] = r \Rightarrow)$ is then equivalent to a purely negative Horn clause.

When one of the conditions of a rule can be immediately solved by syntactic unification, then we can build a *reflexive resolvent.*

$$REF(E) = \{\sigma(c \Rightarrow e) \mid \text{there exists } (c \wedge s = t \Rightarrow e) \in E$$
$$\text{such that } \sigma = mgu(s, t)$$
$$\text{and } \sigma(s = t) \not< \sigma(c), \sigma(e).$$

The inference rules of the unfailing completion procedure can be extended to deal with conditional equalities. *Deduce* adds in E the set of critical pairs obtained by superposition of conditional equalities in E and *Narrow* adds in E the conditional narrowings between two elements. *Reflect* removes a maximal premise of a conditional rule if it is solved. *Delete* and *Trivial* are rules for getting rid of tautologies. *Subsume* allows elimination of redundancies; a rule less general than another one is nonessential and can be deleted. *Simplify* reduces conjectures using conditional rules whose conditions are valid in the underlying theory. We deliberately give a very general format for this rule. For implementation, the search of a proof of the conditions must be bounded in some way. Finally, *Refute* produces a disproof when an empty clause is detected. This can happen only if the initial system contains a purely negative clause.

Deduce:	E	$\vdash E \cup \{c\}$	if $c \in CCP(E)$
Narrow:	E	$\vdash E \cup \{c\}$	if $c \in CN(E)$
Reflect:	tE	$\vdash E \cup \{c\}$	if $c \in REF(E)$
Delete:	$E \cup \{c \Rightarrow s = s\}$	$\vdash E$	
Trivial:	$E \cup \{e \wedge c \Rightarrow e\}$	$\vdash E$	
Subsume:	$E \cup \{C, D\}$	$\vdash E \cup \{C\}$	if $\sigma(C) \subseteq D$
Simplify:	$\cup\{C[\sigma(s)]\}$	$\vdash E \cup \{C[\sigma(t)]\}$	if $\begin{cases} \sigma(s) > \sigma(t) \text{ and} \\ C[\sigma(s)] > \sigma(s = t) \text{ and} \\ E \cup \{C[\sigma(s)]\} \models \sigma(s = t) \end{cases}$
Refute:	$E \cup \{\Rightarrow\}$	$\vdash$ *Disproof.*	

These inference rules are iterated on the initial set of clauses. When this process stops, a system which has the Church-Rosser property on ground terms is guaranteed. This technique is more flexible than previous ones [76, 81], since it does not fail in the presence of nonsimplifying rules or nonorientable equalities.

Moreover, purely negative Horn clauses are also allowed. Note that this method can be viewed as an extension of the recursive approach. A Horn clause can be used to rewrite a term only if its instance by the matching substitution is a simplifying rule. When a Horn clause has no simplifying instance, it is only considered for completion to generate new consequences but it can be deleted once completion is achieved, since it is not needed for normalizing terms. The next example illustrates these notions.

Example 3.3. The following set of Horn clauses, in which $\leq$ denotes the predicate less or equal on integers has the Church-Rosser property on ground terms:

$$
\begin{aligned}
succ(pred(x)) &\rightarrow x \\
pred(succ(x)) &\rightarrow x \\
(0 \leq 0) &\rightarrow true \\
(0 \leq pred(0)) &\rightarrow false \\
(succ(x) \leq y) &\rightarrow (x \leq pred(y)) \\
(pred(x) \leq y) &\rightarrow (x \leq succ(y)) \\
(0 \leq x) = true \Rightarrow (0 \leq succ(x)) &\rightarrow true \\
(0 \leq x) = false \Rightarrow (0 \leq pred(x)) &\rightarrow false \\
(0 \leq pred(x)) = true \Rightarrow (0 \leq x) &\rightarrow true \\
(0 \leq succ(x)) = false \Rightarrow (0 \leq x) &\rightarrow false.
\end{aligned}
$$

The last two clauses have been generated during the completion of the other ones. Note that none of their instances are simplifying: when they are discarded, the remaining system still has the Church-Rosser property.

A Horn clause set can be provided with an initial model, which is often the one of interest. Similarly to equational logic, the clause C is an inductive theorem of a set of Horn clauses S iff S and $S \cup C$ have the same initial model. The notion of inductive reducibility has been extended to this context [95] although it is undecidable [80].

As in [123], an alternative technique is developed in [96] for inductive reasoning in Horn theories. The method in [96] is more general than [123], since the former applies to conditional equalities; it also provides a procedure to compute the instances of the conjecture which are to be reduced. Since it avoids completion, this new technique allows proving much more theorems than inductionless induction.

4. FIRST-ORDER LOGIC WITH EQUALITY

In first-order logic with equality a formula is a set (or a conjunction) of clauses (i.e., disjunction of positive or negative literals). Validity of a formula in the class of all first-order models can be checked by considering only Herbrand models, which are interpretations whose domain is the set of ground terms built from functional symbols [56]. The proof of a formula can proceed by refutation: the formula is first negated and a contradiction is derived by applying inference rules. In 1965,

Robinson proposed *resolution* as a complete inference rule [128]. The resolution method does not deal directly with equality and equality axioms must be added explicitly.

4.1. Paramodulation-Based First-Order Theorem Proving

The replacement of equals by equals has been generalized in [127] from equational logic to first-order logic with equality, in order to improve the efficiency of resolution theorem provers. This new inference rule is called *paramodulation.*

Several simplification inference rules have been designed in order to reduce the size of formulas handled at each step of a deduction. Tautology elimination and subsumption are well known. Term rewriting is also used in the context of first-order logic as a very powerful heuristic, under the name of *demodulation* [144]. It allows one to maintain information in a reduced format. However, until recently, very few theoretical results were known about the effect of simplifying during deduction.

An extension of the classical *semantic trees* method has been designed in [64, 119, 129] to show the refutational completeness of several refinements of the paramodulation rule. The common feature of these refinements is that they allow free interleaving of deduction steps with simplification steps without losing completeness. Due to the importance of simplification in theorem proving, this can be considered as a drastic improvement. In [119] simplifiers must be oriented with a simplification ordering which is isomorphic to ω, but this last restriction is removed in [129, 130], allowing most of the known canonical sets of simplifiers.

It has been observed since 1975 [24, 100, 134] that the computation of critical pairs is a restriction of paramodulation to the case when equalities can be oriented. By extending superposition to nonorientable equalities, a refutationally complete set of inference rules is obtained for first-order logic with equality [129, 130].

Let us denote clauses in sequent form: $L_1,\cdots,L_n \Rightarrow M_1,\cdots,M_m$ represents $\neg L_1 \vee \cdots \vee \neg L_n \vee M_1 \vee \cdots \vee M_m$. Therefore, when we write $\Gamma \Rightarrow \Delta$, Γ and Δ should be understood as sets of positive literals $L, M \ldots$. Let E be a set of clauses. We first give an *ordered* version of the resolution rule [64, 119]. The literals which are resolved are restricted to be maximal within each clause. Furthermore, we assume that they are not equational literals, which will be dealt with later by other inference rules. The set of resolvants of E is:

$$
\begin{aligned}
R(E) = \{\sigma(\Delta, \Lambda \Rightarrow \Gamma, \Pi) \mid & \\
& \text{there exist } (\Delta \Rightarrow L, \Gamma) \in E, (\Lambda, M \Rightarrow \Pi) \in E \\
& \text{such that } \sigma = mgu(L, M) \\
& \text{and } \sigma(L) \not\leq \sigma(\Gamma), \sigma(\Delta), \sigma(\Lambda), \sigma(\Pi)\},
\end{aligned}
$$

where the last line stands for: $\forall M \in \sigma(\Gamma) \cup \sigma(\Delta) \cup \sigma(\Lambda) \cup \sigma(\Pi), \sigma(L) \not\leq M$.

The notion of critical pair is extended from conditional equalities to general clauses by the *right superposition* inference rule. The set $RS(E)$ of right superposi-

tions of E is:

$$RS(E) = \{\sigma(\Delta, \Lambda \Rightarrow l[v] = r, \Gamma, \Pi) \mid$$
$$\text{there exist } (\Delta \Rightarrow l[u'] = r, \Gamma) \in E, (\Lambda \Rightarrow u = v, \Pi) \in E$$
$$\text{such that } \sigma = mgu(u', u)$$
$$\text{and } \sigma(l[u'] = r) \not\leq \sigma(\Gamma), \sigma(\Delta) \text{ and } \sigma(l[u']) \not\leq \sigma(r),$$
$$\text{and } \sigma(u) \not\leq \sigma(v)\}.$$

The notion of conditional narrowing is also extended to general clauses by the *left superposition* inference rule. The set $LS(E)$ of left superpositions of E is:

$$LS(E) = \{\sigma(\Delta, \Lambda, l[v] = r \Rightarrow \Gamma, \Pi) \mid$$
$$\text{there exist } (l[u'] = r, \Delta \Rightarrow \Gamma) \in E, (\Lambda \Rightarrow u = v, \Pi) \in E$$
$$\text{such that } \sigma = mgu(u', u)$$
$$\text{and } \sigma(l[u'] = r) \not\leq \sigma(\Gamma), \sigma(\Delta) \text{ and } \sigma(l[u']) \not\leq \sigma(r),$$
$$\text{and } \sigma(u) \not\leq \sigma(v)\}.$$

Finally, the notion of reflexive resolvants is generalized to the following set:

$$REF(E) = \{\sigma(\Delta \Rightarrow \Pi) \mid \text{there exists } (\Delta, s = t \Rightarrow \Pi) \in E$$
$$\text{such that } \sigma = mgu(s, t)$$
$$\text{and } \sigma(s = t) \not\leq \sigma(\Delta), \sigma(\Pi)\}.$$

The set of deductive rules which are needed for completeness is the following. As mentioned before, *Resolve* only concerns nonequational literals. *Reflect* simulates a resolution step between an inequality and the clause $x = x$. *Deduce*, *Narrow* and *Reflect* are simple extensions of related inference rules for conditional equalities. The positive equalities in the parent clauses which are not involved in a superposition step can be considered as premises of a conditional equality. Also, *Factor* is a new inference rule which merges two unifiable literals of a clause to generate a new clause.

$$\textit{Resolve}: \quad E \vdash E \cup \{c\} \quad \text{if } c \in R(E)$$
$$\textit{Deduce}: \quad E \vdash E \cup \{c\} \quad \text{if } c \in RS(E)$$
$$\textit{Narrow}: \quad E \vdash E \cup \{c\} \quad \text{if } c \in LS(E)$$
$$\textit{Reflect}: \quad E \vdash E \cup \{c\} \quad \text{if } c \in REF(E)$$
$$\textit{Factor}: \quad E \cup \{\Gamma \Rightarrow \Pi, L, L'\} \vdash E \cup \begin{Bmatrix} \Gamma \Rightarrow \Pi, L, L' \\ \sigma(\Gamma \Rightarrow \Pi, L) \end{Bmatrix}$$
$$\text{if } \begin{cases} \sigma = mgu(L, L') \\ \text{and} \\ \sigma(L) \not\leq \sigma(\Gamma), \sigma(\Pi). \end{cases}$$

The rules *Subsume*, *Simplify*, *Delete*, *Refute* of the previous section can also be added with simple modifications [130].

When all the clauses are orientable equalities, the above procedure coincides with completion; when all the clauses are equalities, it coincides with unfailing completion; and when all the clauses are definite Horn clauses, it coincides with

conditional completion. In fact the behaviour of clauses in this approach is very similar to conditional equalities, as noticed in [146]. Given a clause with several positive literals, it can be considered as different conditional equalities according to the positive literal chosen to be the conclusion. For instance, the clause $(e \vee e' \vee \neg c)$ can be considered either as the conditional equality $(c \wedge \neg e' \Rightarrow e)$ or as $(c \wedge \neg e \Rightarrow e')$. If first-order formulas are given as clauses in which atoms and function terms are ordered, they are converted into conditional rewrite rules, and superposition is performed on the maximal literals (this is called *clausal superposition*). This operation is a different formulation of *ordered resolution* and *ordered paramodulation* that were proved refutationally complete in [64]. A variant of the superposition strategy of [129, 130] were proven complete using *model theoretic forcing* [117] for building incrementally models of satisfiable sets of clauses. A similar technique has also been proposed by [12].

Example 4.1. The transitivity of the "less or equal" predicate is proved, assuming the associativity of an operator max which computes the larger of two elements. For simplicity we omit the symbol $\Rightarrow$ for a positive unit clause. The ordering we use to compare terms is lpo with precedence: $\max \prec a \prec b \prec c$. Moreover, any literal with $\leq$ as predicate is assumed to be greater than any equational literal.

$$\Rightarrow (x \leq y), (y \leq x) \tag{1}$$

$$(x \leq y) \Rightarrow max(x, y) = y \tag{2}$$

$$(y \leq x) \Rightarrow max(x, y) = x \tag{3}$$

$$(max(x, y), z) = max(x, max(y, z)) \tag{4}$$

$$(a \leq b) \tag{5}$$

$$(b \leq c) \tag{6}$$

$$(a \leq c) \Rightarrow \tag{7}$$

The skolemized negation of the theorem is given in the last three clauses. The contradiction (i.e., the empty clause) is derived by applying *Resolve*, *Deduce*, *Narrow* and *Factor*. The next consequences are obtained by resolution of (5) and (2), (6) and (2), (1) and (3), (7) and (10):

$$max(a, b) \rightarrow b \tag{8}$$

$$max(b, c) \rightarrow c \tag{9}$$

$$\Rightarrow (x \leq y), max(x, y) \rightarrow y \tag{10}$$

$$max(a, c) \rightarrow a \tag{11}$$

The following consequences are obtained by *Deduce* from (8) into (4), (9) into (12), (11) into (13), (9) into (14):

$$max(a, max(b, z)) \rightarrow max(b, z) \tag{12}$$

$$max(a, c) \rightarrow max(b, c) \tag{13}$$

$$max(b, c) \rightarrow a \tag{14}$$

$$a \rightarrow c \tag{15}$$

Then by *Narrow* from (15) into (7) we derive:

$$(c \leq c) \Rightarrow \tag{16}$$

Last by *Factor* on (1), we get:

$$(x \leq x) \tag{17}$$

The contradiction is straightforward by *Resolve* on (16) and (17). Note also that clauses (14), (15), (16) can be derived by *Simplify* (see Section 3.2). This would be a better strategy, since after a simplification step, one of the parent clauses can be deleted.

Many paramodulation-based theorem provers rely on using equalities as rewrite rules. An implementation of rewriting techniques for first-order theorem proving was undertaken by L. Fribourg in his system SLOG [42]. There, the orientation procedure of equalities does not require a well-founded ordering. However, the so-called *functional reflexive axioms* for equality are needed to ensure completness. Representing clauses as rewrite rules is the basis for a theorem-prover for Horn clauses which uses a unit strategy and is described in [118].

4.2. Boolean Ring-Based First-Order Theorem Proving

Another term rewriting approach transforms first-order predicate logic into a special case of equational logic. This transformation is done through a canonical system for Boolean algebra, using + (*exclusive-or*) and * (*and*) as the logical connectives (also 0 for *false* and 1 for *true*). The mathematical structure of this representation is called *Boolean ring* [138]. Boolean rings yield a unique normal form for every Boolean term that can be obtained via rewriting with a class rewrite system, which also provides a decision procedure for the propositional calculus [61]. When applied to first-order logic, one first converts the negation of the sentences into rewrite rules, then applies superpositions and reductions as in the other completion procedures until $1 = 0$ is generated, which indicates a refutational proof, or until there are no more critical pairs left. In [59] a strategy, called the *N-strategy*, which restricts the type of necessary critical pairs was presented. The N-strategy was extended to deal with first-order logic with equality in [60] and was further refined in [109].

The N-strategy was implemented in the system TeRSe [62]. More recently, a variant of N-strategy has been developed in the system THEOPOGLES [109], where the critical pair generation process is improved by a simpler superposition algorithm.

A related method, inspired by the Gröbner basis generation in polynomial rings [26], was presented in [84] and implemented in the system RRL. The completeness of a refined procedure was proved in [8].

4.3. Equational Matings

Another related method for first-order theorem proving with equality is *equational mating*. The method of matings, due to Andrews [3] and also investigated by Bibel [16], was motivated by two considerations: to avoid breaking a formula into parts, and to avoid transforming it in clausal form. Formulas are instead kept in negative

normal form. Through such a quantifier-free formula, vertical paths can be threaded which are sets of literals obtained by going down the formula's syntactic structure, merging subpaths at conjunctions and choosing subpaths at disjunctions. The method relies on the fact that a quantifier-free formula in negative normal form is unsatisfiable if and only if all its vertical paths are unsatisfiable. It is an incremental refutation procedure that attempts to close all vertical paths with a common substitution, then, in case of failure, splits quantified formulae and iterates. This is where the concept of matings comes in. In the context of Horn clauses, a mating is a set of pairs of literals of opposite signs spanning all vertical paths, such that all these pairs are globally unified by some substitution. Finding a mating signifies that its unifying substitution closes all vertical paths. The method of matings was extended to first-order languages with equality in [47, 48]. This sound and complete extension requires the notion of *equational matings*. An equational mating is a set of mated sets, where a mated set consists of several positive equations and a single negated equation. Unification is replaced by rigid equational unification.

5. CONCLUSION AND FUTURE WORK

Application of term rewriting in computer science is not confined to automated deduction. Term rewriting systems also provided operational semantics of several programming languages [31, 53, 58, 91, 115]. Lambda calculus and combinatory logic are also worth mentioning especially since the Church-Rosser theorem for lambda calculus motivated the study of term rewriting in the first place [13].

It is interesting to note that although rewriting is such a powerful tool for theorem proving, it was not designed for this purpose. The Knuth-Bendix completion procedure was originally intended for generating canonical sets of rewrite rules rather than for proving any specific equational theorem. Thus, the emphasis in the design of the inference system was on resolving all possible nonconfluent critical pairs. In a typical theorem proving application, on the other hand, critical pair generation should also take into consideration the specific theorem being proved. In other words, the critical pairs which may lead to an eventual proof of the target theorem should be considered first, and those which cannot be part of any such proof should be considered as redundant and discarded. An example of a goal-oriented heuristic for proving individual theorems in unfailing completion was described in [1]. An abstract framework of completion procedure with targets is proposed in [18], where specific issues such as fairness and redundancy were discussed. However, we feel that much more work needs to be done in this area.

Other promising directions for rewriting techniques should be mentioned. If higher-order functions are allowed, the expressive power of functional languages can be increased considerably. The combination of first-order rewriting with high-order polymorphic computation was shown feasible by a recent work, stating that the Church-Rosser property is preserved when a term rewriting system is extended by the typed lambda-calculus [45].

Inspired by constraint logic programming, a notion of constrained rewriting also emerged. The concept of constrained deduction and constrained rewriting was proposed in [27, 92]. The intention is to add more deductive power to the unification mechanism which is the basis of simplification and deduction rules. This

enhancement is obtained by keeping more information about variables and by using this information in a tractable way. It also includes inference rules for interleaving first-order deduction rules and specific constraint-solving mechanisms. Thus, constrained deduction has more expressive power and permits a more flexible scheduling of inference mechanisms than deduction without constraints. Theorem-proving methods based on these ideas are being developed.

We thank the referees for their constructive remarks and suggestions and the research group Eureca for its stimulating environment.

REFERENCES

1. Anantharaman, S., and Andrianarivelo, A., Heuristical Critical Pair Criteria in Automated Theorem Proving, *Proceedings of DISCO'90*, Capri (Italy) 1990, pp. 184–193.
2. Anantharaman, S., and Hsiang, J., An Automated Proof of the Moufang Identities in Alternative Rings, *Automated Reasoning* 6:79–109 (1990).
3. Andrews, P. B., Theorem Proving via General Matings, *J. ACM* 28(2):193–214 (1981).
4. Avenhaus, J., Göbel, R., Gramlich, B., Madlener, K., and Steinbach, J., TRSPEC: A Term Rewriting Based System for Algebraic Specifications, in: *Proceedings of the 1st International Workshop on Conditional Term Rewriting Systems*, Orsay, France, 1987, pp. 245–248.
5. Bachmair, L., Proof by Consistency in Equational Theories, in: *3rd Symposium on Logic in Computer Science*, Edinburgh, Scotland, 1988, pp. 228–233.
6. Bachmair, L., *Canonical Equational Proofs*, Computer Science Logic, Progress in Theoretical Computer Science, Birkhäuser Verlag, 1991.
7. Bachmair, L., and Dershowitz, N., Commutation, Transformation and Termination, in: *Proceedings of the 8th International Conference on Automated Deduction, Oxford U.K.*, 1986, pp. 5–20.
8. Bachmair, L., and Dershowitz, N., Inference Rules for Rewrite-Based First-Order Theorem Proving, in: *Proceedings of the Second Symposium on Logic in Computer Science*, Ithaca, N.Y., 1987, pp. 331–337.
9. Bachmair, L., and Dershowitz, N., Completion for Rewriting Modulo a Congruence, *Theoret. Comput. Sci.*, 67(2–3):173–202 (1989).
10. Bachmair, L., Dershowitz, N., and Hsiang, J., Orderings for Equational Proofs, in: *Proceedings of the First Symposium on Logic in Computer Science*, Boston, Mass., 1986, pp. 346–357.
11. Bachmair, L., Dershowitz, N., and Plaisted, D., Completion without Failure, in: H. Aït-Kaci and M. Nivat (eds.), *Resolution of Equations in Algebraic Structures*, Vol. 2, Academic Press, San Diego, Calif., 1989, pp. 1–30.
12. Bachmair, L., and Ganzinger, H., On Restriction of Ordered Paramodulations with Simplification, in: *Proceedings of the 10th International Conference on Automated Deduction*, Kaiserslautern (Germany) 1990, pp. 427–441.
13. Barendregt, H., *The Lambda-Calculus, Its Syntax and Semantics*, 2nd Ed., North-Holland, Amsterdam, 1984.
14. Bellegarde, F., and Lescanne, P., Transformation Orderings, in: *12th Colloqium on Trees in Algebra and Programming, TAPSOFT*, Pisa (Italy) 1987, pp. 69–80.
15. Bergstra, A., Klop, J. W., Conditional Rewrite Rules: Confluency and Termination, *J. Comput. Syst. Sci.*, 32(3):323–362 (1986).
16. Bibel, W., *Automated Theorem Proving*, Friedrich Viewig & Sohn, Braunschweig, 1982.
17. Birkhoff, G., On the Structure of Abstract Algebras, in: *Proceedings of the Cambridge Phil. Soc.* 31:433–454 (1935).

18. Bonacina, M. P., and Hsiang, J., Completion Procedures as a Semidecision Procedure, in: *Proceedings of the 2nd Workshop on Conditional and Typed Rewriting Systems*, Montreal (Canada) 1990, pp. 206–232.
19. Book, R. V., Thue Systems as Rewriting Systems, *J. Symbolic Comput.* 3(1–2):39–68 (1987).
20. Bousdira, W., *Etude des Propriétés des Systèmes de Réécriture Conditionnelle: Mise en Œuvre d'un Algorithme de Complétion*, Ph.D. Dissertation, Institut National Polytechnique de Lorraine, France, 1990.
21. Bousdira, W., and Rémy, J. L., Reveur4: A Laboratory for Conditional Rewriting, in: *Proceedings of the 4th Symposium on Theoretical Aspects of Computer Science*, 1987, pp. 472–473.
22. Boyer, R. S., and Moore, J. S., A *Computational Logic*, Academic Press, New York, 1979.
23. Brand, D., Darringer, J. A., and Joyner, W. H., Completeness of Conditional Reductions, Report RC 07404, IBM USA Research Center, Yorktown Heights, N.Y., 1978.
24. Brown, T., *A Structured Design-Method for Specialized Proof Procedures*, Ph.D. Dissertation, California Institute of Technology, Pasadena, 1975.
25. Buchberger, B., Ein algorithmus zum auffiden der basiselemente des restklassenringes nach einem nulldimensionalen polynomideal, Ph.D. Dissertation, University of Innsbruck, Austria, 1965.
26. Buchberger, B., History and Basic Features of the Critical-Pair/Completion Approach, *J. Symbolic Comput.*, 3(1/2):3–38 (1987).
27. Bürckert, H.-J., A Resolution Principle for Clauses with Constraints, in: *Proceedings of the 10th International Conference on Automated Deduction*, Kaiserslautern, Germany, 1990, pp. 178–192.
28. Ben Cherifa, A., and Lescanne, P., Termination of Rewriting Systems by Polynomial Interpretations and Its Implementation, *Sci. Comput. Programming* 9(2):137–160 (1987).
29. Comon, H., An Effective Method for Handling Initial Algebras, in: *Proceedings of the International Workshop on Algebraic and Logic Programming*, Gaussig (Germany), 1988, pp. 108–118.
30. Dershowitz, N., Orderings for Term-Rewriting Systems, *Theoret. Comput. Sci.* 17:279–301 (1982).
31. Dershowitz, N., Computing with Rewrite Systems, *Information and Control*, 65(2/3):122–157 (1985).
32. Dershowitz, N., Corrigendum to Termination of Rewriting, *J. Symbolic Comput.* 4:409–410 (1987).
33. Dershowitz, N., Termination of Rewriting, *J. Symbolic Comput.* 3(1/2):69–116 (1987).
34. Dershowitz, N., Completion and Its Applications, in: H. Aït-Kaci and M. Nivat (eds.), *Resolution of Equations in Algebraic Structures*, Vol. 2, Academic Press, San Diego, Calif., 1989, pp. 31–86.
35. Dershowitz, N., and Okada, M., A Rationale for Conditional Equational Programming, *Theoretical Comput. Sci.*, 75:111–138 (1990).
36. Dershowitz, N., and Sivakumar, G., Goal-Directed Equation Solving, in: *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minn., 1988, pp. 166–170.
37. Dick, A. J. J., ERIL Equational Reasoning: An Interactive Laboratory, in: *Proceedings of the EUROCAL Conference*, Linz, Austria, 1985.
38. Evans, T., On Multiplicative Systems Defined by Generators and Relations, in: *Proceedings of the Cambridge Phil. Soc.*, 67:637–649 (1951).
39. Fages, F., Le Système KB. Manuel de Référence, Présentation et Bibliographie, Mise en Œuvre, Technical report, Greco de Programmation, Bordeaux, France, 1984.
40. Fay, M., First-Order Unification in Equational Theories, in: *Proceedings of the 4th Workshop on Automated Deduction*, Austin, Tex., 1979, pp. 161–167.
41. Forgaard, R., and Guttag, J., REVE: A Term Rewriting System Generator with Failure-Resistant Knuth-Bendix, Technical report, MIT-LCS, 1984.

42. Fribourg, L., SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting, in: *Proceedings of the IEEE Symposium on Logic Programming*, Boston, Mass., 1985, pp. 172–184.
43. Fribourg, L., A Superposition Oriented Theorem Prover, *Theoret. Comput. Sci.*, 35:129–164 (1985).
44. Fribourg, L., A Strong Restriction of the Inductive Completion Procedure, *J. Symbolic Comput.* 8(3):253–276 (1989).
45. Gallier, J., and Breazu-Tannen, V., Polymorphic Rewriting Conserves Algebraic Strong Normalization and Confluence, in: *16th Colloquium on Automata, Languages and Programming*, Stresa (Italy) 1989, pp. 137–150.
46. Gallier, J., Narendran, P., Plaisted, D., and Snyder, W., Rigid E-Unification: NP-Completeness and Applications to Theorem Proving, *Inform. Comput.* 87(1–2):129–195 (1990).
47. Gallier, J., Narendran, P., Raatz, S., and Snyder, W., Theorem Proving Using Equational Matings and Rigid E-Unification, Technical report, University of Pennsylvania, N.S.F., 1988.
48. Gallier, J., Raatz, S., and Synder, W., Rigid E-Unification and Its Applications to Equational Matings, in: H. Aït-Kaci and M. Nivat (eds.), *Resolution of Equations in Algebraic Structures*, vol. 1, Academic Press, San Diego, Calif., 1989, pp. 151–216.
49. Gallier, J., and Snyder, W., Complete Sets of Transformations for General E-Unification, *Theoret. Comput. Sci.* 67(2–3):203–260 (1989).
50. Ganzinger, H., A Completion Procedure for Conditional Equations, in: *Proceedings of the 1st International Workshop on Conditional Term Rewriting Systems*, Orsay, France, 1987.
51. Garland, S. J., and Guttag, J. V., An Overview of LP, the Larch Prover, in: *Proceedings of the 3rd Conference on Rewriting Techniques and Applications*, Chapel Hill, N.C., 1989, pp. 137–151.
52. Goguen, J. A., How to Prove Algebraic Inductive Hypotheses without Induction, with Applications to the Correctness of Data Type Implementation, in: *Proceedings of the 5th International Conference on Automated Deduction*, Les Arcs, France, 1980, pp. 356–373.
53. Goguen, J. A., Jouannaud, J.-P., and Meseguer, J., Operational Semantics for Order-Sorted Algebra, in: *Proceedings of the 12th International Colloquium on Automata, Languages and Programming*, Nafplion, Greece, 1985, pp. 221–231.
54. Grätzer, G., *Universal Algebra*, 2nd Ed., Springer-Verlag, New York, 1979.
55. Henkin, L., The Logic of Equality, *Am. Math. Monthly* 84(8):597–612 (1977).
56. Herbrand, J., Recherches sur la Théorie de la Démonstration, *Travaux de la Soc. des Sciences et des Lettres de Varsovie, Classe III* 33/128 (1930).
57. Hermann, M., Kirchner, C., and Kirchner, H., Implementations of Term Rewriting Systems, *The Computer Journal, British Comput. Soc.* 34(1):20–33 (1991).
58. Hoffmann, C. M. and O'Donnell, M. J., Programming with Equations, *Transactions on Programming Languages and Systems* 4(1):83–112 (1982).
59. Hsiang, J., Refutational Theorem Proving Using Term Rewriting Systems, *Artif. Intell.* 25(1):255–300 (1985).
60. Hsiang, J., Rewrite Methods for Theorem Proving in First Order Theory with Equality, *J. Symbolic Comput.* 3(1/2):133–151 (1987).
61. Hsiang, J., and Dershowitz, N., Rewrite Methods for Clausal and Non-Clausal Theorem Proving, in: *Proceedings of the 10th International Colloquium on Automata, Languages and Programming*, Barcelona, Spain, 1983, pp. 331–346.
62. Hsiang, J., and Josephson, N. A. TeRSe: A Term Rewriting Theorem Prover, in: *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, General Electric, Corporate Research and Development, Schenectady, N.Y., 1984, pp. 229–243.
63. Hsiang, J., and Mzali, J., SbREVE user's guide, Technical report, LRI, Orsay, France, 1988.

64. Hsiang, J., and Rusinowitch, M., A New Method for Establishing Refutational Completeness in Theorem Proving, in: *Proceedings of the 8th International Conference on Automated Deduction*, Oxford, U.K., 1986, pp. 141–152.

65. Hsiang, J., and Rusinowitch, M., On Word Problem in Equational Theories, in: *Proceedings of the 14th International Colloquium on Automata, Languages and Programming*, Karlsruhe, Germany, 1987, pp. 54–71.

66. Huet, G., Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, *J. AMC* 27(4):797–821 (1980).

67. Huet, G., A Complete Proof of Correctness of the Knuth and Bendix Completion Algorithm, *J. Comput. Systems and Sciences* 23:11–21 (1981).

68. Huet, G., and Hullot, J.-M., Proofs by Induction in Equational Theories with Constructors, *J. Comput. Syst. Sci.*, 25(2):239–266 (1982).

69. Hullot, J.-M., Canonical Forms and Unification, in: *Proceedings of the 5th International Conference on Automated Deduction*, Les Arcs, France, 1980, pp. 318–334.

70. Jouannaud, J.-P., Confluent and Coherent Equational Term Rewriting Systems: Applications to Proofs in Abstract Data Types, in: *Proceedings of the 8th Colloquium on Trees in Algebra and Programming*, L'Aquila, Italy, 1983, pp. 269–283.

71. Jouannaud, J.-P., and Kirchner, C., Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification, in: Computational Logic. Essays in honor of Alan Robinson, MIT Press, chapter 8, pp. 257–321, (1991).

72. Jouannaud, J.-P., Kirchner, C., and Kirchner, H., Incremental Construction of Unification Algortihms in Equational Theories, in: *Proceedings of the 10th International Colloquium on Automata, Languages and Programming*, Barcelona, Spain, 1983, pp. 361–373.

73. Jouannaud, J.-P., and Kirchner, H., Completion of a Set of Rules Modulo a Set of Equations, *SIAM J. Comput.* 15(4):1155–1194 (1986).

74. Jouannaud, J.-P., and Kounalis, E. Automatic Proofs by Induction in Theories without Constructors, *Inform. Comput.* 82:1–33 (1989).

75. Jouannaud, J.-P., Lescanne, P., and Reinig, F., Recursive Decomposition Ordering, in: Bjørner D. (ed.), *Formal Description of Programming Concepts 2*, North-Holland, Garmisch-Partenkirchen, Germany, 1982, pp. 331–348.

76. Jouannaud, J.-P., and Waldmann, B., Reductive Conditional Term Rewriting Systems, in: *3rd IFIP Conference on Formal Description of Programming Concepts*, Ebberup, Denmark, 1986.

77. Kamin, S., and Lévy, J.-J., Attempts for Generalizing the Recursive Path Ordering, INRIA, Rocquencourt, France, 1982.

78. Kaplan, S., Conditional Rewrite Rules, *Theoret. Comput. Sci.* 33:175–193 (1984).

79. Kaplan, S., Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence, *J. Symbolic Comput.* 4(3):295–334 (1987).

80. Kaplan, S., and Choquer, M., On the Decidability of Quasi-Reducibility, *EATCS Bulletin* 28:32–34 (1986).

81. Kaplan, S., and Rémy, J.-L., Completion Algorithms for Conditional Rewriting Systems, in: H. Aït-Kaci and M. Nivat (eds.), *Resolution of Equations in Algebraic Structures*, Vol. 2, Academic Press, San Diego, Calif. 1989, pp. 141–170.

82. Kapur, D., and Musser, D. R., Proof by Consistency, *Artif. Intell.* 31(2):125–157 (1987).

83. Kapur, D., Musser, D. R., and Narendran, P., Only Prime Superpositions Need to be Considered in the Knuth-Bendix Completion Procedure, *J. Symbolic Comput.* 6(2):19–36 (1988).

84. Kapur, D. and Narendran, P., An Equational Approach to Theorem Proving in First-Order Predicate Calculus, in: *Proceedings of the International Joint Conference on Artificial Intelligence*, Los Angeles, Calif., 1985.

85. Kapur, D., Narendran, P., and Zhang, H., Proof by Induction Using Test Sets, in: *Proceedings of the 8th International Conference on Automated Deduction*, Oxford, U.K., 1986, pp. 99–117.

86. Kapur, D., Narendran, P., and Zhang, H., On Sufficient Completeness and Related Properties of Term Rewriting Systems, *Acta Inform.* 24:395–415 (1987).

87. Kapur, D., and Zhang, H., RRL: A Rewrite Rule Laboratory, in: *Proceedings of the 9th International Conference on Automated Deduction*, Argonne, Ill., 1988, pp. 768–769.

88. Kirchner, C., Méthodes et outils de conception systématique d'algorithmes d'unification dans les théories équationnelles, Ph.D. Dissertation, Université de Nancy I, France, 1985.

89. Kirchner, C., Computing Unification Algorithms, in: *Proceeding of the First Symposium on Logic in Computer Science*, Boston, Mass., 1986, pp. 206–216.

90. Kirchner, C., and Kirchner, H., Reveur-3: Implementation of a General Completion Procedure Parametrized by Built-In Theories and Strategies, *Sci. Comput. Programming* 20(8):69–86 (1986).

91. Kirchner, C., Kirchner, H., and Meseguer, J., Operational Semantics of OBJ-3, in: *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, Tampere (Finland) 1988, pp. 287–301.

92. Kirchner, C., Kirchner, H., and Rusinowitch, M., Deduction with Symbolic Constraints, *Revue d'Intelligence Artificelle* 4(3):9–52 (1990).

93. Knuth, D. E., and Bendix, P. B., Simple Word Problems in Universal Algebras, in: J. Leech (ed.), *Comput. Prob. Abstract Algebra*, Pergamon Press, Oxford, U.K., 1970, pp. 263–297.

94. Kounalis, E., Completeness in Data Type Specifications, in: *Proceedings EUROCAL Conference*, Linz, Austria, 1985, pp. 348–362.

95. Kounalis, E., and Rusinowitch, M., On Word Problem in Horn Logic, *J. Symbolic Comput.*, 11(1/2) pp. 113–127, (1991).

96. Kounalis, E., and Rusinowitch, M., Mechanizing Inductive Reasoning, in: *Proceedings of the AAAI Conference*, Boston, Mass., 1990, pp. 240–245.

97. Kucherov, G. A., A New Quasi-Reducibility Testing Algorithm and Its Application to Proofs by Induction, in: *Proceedings of the International Workshop on Algebraic and Logic Programming*, Gaussig (Germany) 1988, pp. 204–213.

98. Küchlin, W., A Confluence Criterion Based on the Generalized Knuth-Bendix Algorithm, in: *Proceedings of the EUROCAL Conference*, Linz, Austria, 1985, pp. 390–399.

99. Küchlin, W., Inductive Completion by Ground Proof Transformation, in: H. Aït-Kaci and M. Nivat (eds.), *Resolution of Equations in Algebraic Structures*, Vol. 2, Academic Press, San Diego, Calif., 1989, pp. 211–244.

100. Lankford, D. S., Canonical Inference, Technical Report, Louisiana Technical University, Ruston, 1975.

101. Lankford, D. S., On Deciding Word Problems by Rewrite Rules Simplifiers, Technical Report, Louisiana Technical University, Ruston, 1977.

102. Lankford, D. S., On Proving Term Rewriting Systems Are Noetherian, Technical Report, Louisiana Technical University, Mathematics Department, Ruston, 1979.

103. Lankford, D. S., A Simple Explanation of Inductionless Induction, Memo MTP-14, Louisiana Technical University, Department of Mathematics, Ruston, 1981.

104. Lankford, D. S., and Ballantyne, A., Decision Procedures for Simple Equational Theories with Associative Commutative Axioms: Complete Sets of Associative Commutative Reductions, Technical Report, University of Texas at Austin, Department of Mathematics and Computer Science, 1977.

105. Lazrek, A., Lescanne, P., and Thiel, J.-J., Tools for Proving Inductive Equalities, Relative Completeness, and ω-Completeness, *Inform. Comput.* 84(1):47–70 (1990).

106. Lescanne, P., Computer Experiments with the REVE Term Rewriting Systems Generator, in: *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*, Austin, Texas, 1983, pp. 99–108.

107. Lescanne, P., Completion Procedures as Transition Rules + Control, in: *TAPSOFT '89*, Barcelona, Spain, 1989, pp. 28–41.

108. Lescanne, P., On the Recursive Decomposition Ordering with Lexicographical Status and Other Related Orderings, *J. Automated Reasoning* 6:39–49 (1990).

109. Müller, J., THEOPOGLES, A Theorem Prover Based on First-Order Polynomials and a Special Knuth-Bendix Procedure, in: *Proceedings of the 11th German Workshop on Artificial Intelligence*, Geseke (Germany) 1987.

110. Musser, D. R., Abstract Data Type Specification in the AFFIRM System, *IEEE Trans. Software Eng.*, 6(1) (1980).

111. Musser, D. R., On Proving Inductive Properties of Abstract Data Types, in: *Proceedings of the 7th ACM Symposium on Principles of Programming Languages*, Las Vegas, 1980, pp. 154–162.

112. Mzali, J., Méthodes de filtrage équationnel et de preuve automatique de théorèmes, Ph.D. Dissertation, Université de Nancy I, France, 1986.

113. Newman, M. H. A., On Theories with a Combinatorial Definition of Equivalence, *Ann. Math.*, 43:223–243 (1942).

114. Nutt, Réty, P., and Smolka, G., Basic Narrowing Revisited, *J. Symbolic Comput.* 7(3/4):295–318 (1989).

115. O'Donnell, M. J., *Equational Logic as a Programming Language*, MIT Press, Cambridge, Mass., 1985.

116. Ohsuga, A., and Sakai, K., METIS, A Term Rewriting System Generator, in: *RIMS Symposium on Software Science and Engineering*, Kyoto (Japan) 1986.

117. Pais, J., and Peterson, G. E., Using Forcing to Prove Completeness of Resolution and Paramodulation, *J. Symbolic Comput.*, 11(1/2):3–19 (1991).

118. Paul, E., On Solving the Equality Problem in Theories Defined by Horn Clauses, *Theoret. Comput. Sci.* 44(2):127–153 (1986).

119. Peterson, G., A Technique for Establishing Completeness Results in Theorem Proving with Equality, *SIAM J. Comput.* 12(1):82–100 (1983).

120. Peterson, G., and Stickel, M., Complete Sets of Reductions for Some Equational Theories, *J. ACM*, 28:233–264 (1981).

121. Plaisted, D., Semantic Confluence and Completion Method, *Inform. Contr.* 65:182–215 (1985).

122. Plotkin, G., Building-In Equational Theories, *Mach. Intell.* 7:73–90 (1972).

123. Reddy, U., Term Rewriting Induction, in: *Proceedings of the 10th International Conference on Automated Deduction*, Kaiserslautern, Germany, 1990, pp. 162–177.

124. Rémy, J.-L., Etude des systèmes de réécriture conditionnels et applications aux types abstraits algébriques, Ph.D. Dissertation, Institut National Polytechnique de Lorraine, Nancy, France, 1982.

125. Réty, P., Improving Basic Narrowing, in: *Proceedings of the 2nd Conference on Rewriting Techniques and Applications*, Bourdeaux, France, 1987, pp. 228–241.

126. Réty, P., Kirchner, C., Kirchner, H., and Lescanne, P., Narrower: A New Algorithm for Unification and Its Application to Logic Programming, in: *Proceedings of the 1st Conference on Rewriting Techniques and Applications*, Dijon, France, 1985, pp. 141–157.

127. Robinson, G., and Wos, L. T., Paramodulation and First-Order Theorem Proving, in: B. Meltzer and D. Mitchie (eds.), *Machine Intelligence 4*, Edinburgh University Press, U.K., 1969, pp. 135–150.

128. Robinson, J. A., A Machine-Oriented Logic Based on the Resolution Principle, *J. ACM*, 12:23–41, 1965.

129. Rusinowitch, M., Démonstration Automatique par des Techniques de Réécriture, Ph.D. Dissertation, Université de Nancy I, France, 1987.

130. Rusinowitch, M., Theorem-Proving with Resolution and Superposition: *J. Symbolic Computation M* (1/2) pp. 21–69, (1991).

131. Schoenfield, J. R., *Mathematical Logic*, Addison-Wesley, Reading, Mass., 1970.

132. Selman, A., Completeness of Calculii for Axiomatically Defined Classes of Algebras, *Algebra Universalis* 2:20–32 (1972).

133. Siekmann, J., Unification Theory, *J. Symbolic Comput.* 7(3/4):207–274 (1989).
134. Slagle, J. R., Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity, *J. ACM* 21(4):622–642 (1974).
135. Snyder, W., *Complete Sets of Transformations for General Unification*, Ph.D. Dissertation, University of Pennsylvania, 1988.
136. Stickel, M. E., A Unification Algorithm for Associative-Commutative Functions, *J. ACM* 28:423–434 (1981).
137. Stickel, M. E., A Case Study of Theorem Proving by the Knuth-Bendix Method: Discovering that $x^3 = x$ Implies Ring Commutativity, in: *Proceedings of the 7th International Conference on Automated Deduction*, Napa Valley, Calif., 1984, pp. 248–258.
138. Stone, M., The Theory of Representations for Boolean Algebra, *Trans. AMS* 40:37–111 (1936).
139. Tarski, A., Equational Logic and Equational Theories of Algebras, in: K. Schütte, (ed.), *Contributions to Mathematical Logic*, North-Holland, Amsterdam, 1968, pp. 275–288.
140. Taylor, W., Equational Logic, *Houston J. Math.*, vol. 13. Also in [86] (1979).
141. Thiel, J.-J., Stop Losing Sleep over Incomplete Data Type Specifications, in: *Proceedings of the 11th ACM Symposium on Principles of Programming Languages*, Salt Lake City, UT, 1984, pp. 76–82.
142. Winkler, F., Knuth-Bendix Procedure and Buchberger Algorithm—A Synthesis, in: *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation*, Portland, Oreg., 1989, pp. 55–67.
143. Winkler, F., and Buchberger, B., A Criterion for Eliminating Unnecessary Reductions in the Knuth-Bendix Algorithm, in: *Proceedings of the Colloquium on Algebra, Combinatorics and Logic in Computer Science*, Györ, Hungary, 1983.
144. Wos, L., Robinson, G. A., Carso, D. F., and Shalla, L., The Concept of Demodulation in Theorem Proving, *J. ACM* 14(4):698–709 (1967).
145. You, J.-H., Enumerating Outer Narrowing Derivations for Constructor-Based Term Rewriting Systems, *J. Symbolic Comput.* 7(3/4):319–342 (1989).
146. Zhang, H., and Kapur, D., First-Order Theorem Proving Using Conditional Rewrite Rules, in: *Proceedings of the 9th International Conference on Automated Deduction*, Argonne, Ill., 1988, pp. 1–20.
147. Zhang, H., and Kapur, D., Consider Only General Superpositions in Completion Procedures, in: *Proceedings of the 3rd Conference on Rewriting Techniques and Applications*, Chapel Hill, N.C., 1989, pp. 513–527.
148. Zhang, H., Kapur, D., and Krishnamoorthy, M. S., A Mechanizable Induction Principle for Equational Specifications, in: *Proceedings of the 9th International Conference on Automated Deduction*, Argonne, Ill., 1988, pp. 162–181.